

PERSISTENT ARCHIVES

This application claims the benefit of U.S. Provisional Application No. 60/191,662, filed March 23, 2000, U.S. Provisional Application No. 60/255,795, filed
5 December 15, 2000, U.S. Provisional Application No. 60/255,794, filed December 15, 2000, and U.S. Provisional Application No. __/__, Howrey Dkt. No. 02737.0007.PZUS01, entitled "PERSISTENT ARCHIVES AND KNOWLEDGE-BASE PERSISTENT ARCHIVES," filed March 5, 2001, all of which are hereby
10 fully incorporated by reference herein as though set forth in full. The U.S. Government has a paid-up license in this invention and the right in limited circumstances to require the patent owner to license others on reasonable terms as provided for by the terms of contract number F19628-96-C-0020 awarded by the Department of Defense.

BACKGROUND OF THE INVENTION

15 1. Field of the Invention.

This invention relates generally to the field of data archives, and, more specifically, persistent archives of collections of data objects.

2. Related Art.

The long-term storage and access of digital information is a major challenge.
20 The rapid change of technology resulting in obsolescence of archival storage media and database management systems, coupled with the very large volumes of data (terabytes to petabytes in size) involved, appears to make the problem intractable. A concern is that, when access to data in the archive is desired at some point in the future, the technology used to create the archive may be obsolete and unavailable, and
25 the technology existing in the future may not allow access to the data absent time-

consuming conversion efforts. Another concern is that the data may not be understandable due to the passage of time and loss of context.

SUMMARY

5 The invention provides a persistent archive of a collection of data objects tangibly embodied on a processor readable medium. The persistent archive comprises a self-describing, infrastructure-independent representation of a logical structure for the collection, and a self-describing, infrastructure-independent representation of the data objects.

10 For purposes of this disclosure, the phrase “self-describing” is a flexible concept which varies according to the circumstances, but it is generally used to refer to an element whose meaning is apparent from the element itself or through resort to no more than generally understood principles; the term “persistent” generally means the quality or capability of being accessible and usable at an indefinite point in time in
15 the future; and the phrase “infrastructure-independent” generally refers to the state or quality of being independent of a particular storage or computing platform or implementation or at most limited to only a generic class of storage or computing platforms or implementations.

20 Since the elements of the persistent archive--the logical structure of the collection and the data objects--are expressed in a self-describing, infrastructure-independent form, the collection can be re-instantiated and understood at an indefinite point in time in the future no matter what the specific state of technology is at the time. Consequently, the archive is persistent.

25 The logical structure of the collection may be expressed through a variety of means, but, in one example, the logical structure is expressed in the form of an

eXtensible Markup Language (XML) Document Type Definition (DTD), which defines elements of data objects or collections, their interrelationship, and their attributes. Since an XML DTD is a non-proprietary and widely known mode of expression, is platform-independent, and is emerging as a standard, it qualifies as a
5 self-describing infrastructure-independent means of expressing the logical structure of the collection.

In another example, the logical structure is expressed in the form of Structured Query Language (SQL) commands for creating relational database tables. Since SQL is a non-proprietary and widely known mode of expression, and is platform
10 independent, this mode of expression also qualifies as self-describing.

The data objects of the collection may also be expressed through a variety of means, but, in one example, the data objects are expressed in the form of tagged XML data objects, in which components of the data objects are tagged with element or attribute names from the DTD. Since the DTD defines the meaning and
15 interrelationship of the elements and attributes, the tagging, in associating element and attribute names with components of the data objects, qualifies as a self-describing, infrastructure-independent form of expression.

The persistent archive may also include a self-describing, infrastructure-independent representation of a presentation mechanism for one or more of the data
20 objects. The presentation mechanism may capture the “look and feel” of certain presentation formats for the data objects that may be re-created at some point in the future when the archive is re-instantiated.

In one example, the presentation mechanism is represented in the form of an eXtensible Stylesheet Language (XSL) style sheet which specifies one or more
25 templates for transforming XML-tagged data objects into desired presentation entities,

such as a HTML page for presentation on a web browser. Since XSL is written in XML, it as well qualifies as a self-describing, infrastructure-independent form of expression.

5 The invention also provides a method of ingesting data objects into the persistent archive, a method of instantiating the persistent archive as a query-able mechanism, a method of migrating the archive to a new medium, and a method of presenting the data objects using a self-describing, infrastructure-independent representation of a presentation mechanism stored with the archive.

10 One embodiment of the method of ingesting data objects into the archive comprises the steps of transforming a representation of the data objects into a self-describing, infrastructure-independent representation of the data objects, and then archiving the self-describing, infrastructure-independent representation of the data objects with a self-describing, infrastructure-independent representation of a logical structure for the collection.

15 One embodiment of the method of instantiating the persistent archive comprises the steps of retrieving from the persistent archive a self-describing, infrastructure-independent representation of a logical structure for the collection, creating on a medium a query-able mechanism in accordance with the logical structure, retrieving from the archive a self-describing, infrastructure-independent
20 representation of one or more data objects, and then loading the one or more data objects into the query-mechanism.

In one example, the query-able mechanism is a database management system. The data objects, once instantiated on the database management system, may be rapidly accessed using database queries. The retrieved objects may then be presented
25 using a presentation mechanism retrieved from the archive. In one example, the

presentation mechanism is an HTML web page which specifies the format for displaying data objects on a web browser.

One embodiment of the method of migrating a persistent archive to a new medium comprises retrieving the persistent archive from a first medium, optionally
5 redefining the logical structure of the collection or the self-describing, infrastructure-independent representation of the data objects in the archive, and storing the persistent archive as optionally redefined onto a second medium.

One embodiment of the method of presenting one or more data objects from the persistent archive comprises retrieving from the archive a self-describing,
10 infrastructure-independent representation of a presentation mechanism for the one or more data objects, and presenting the one or more data objects using the presentation mechanism.

A system for maintaining a persistent archive is also provided. In one embodiment, the system comprises an ingestion subsystem for ingesting objects into the archive; and an instantiation subsystem for instantiating the archive onto a query-
15 able mechanism. The instantiation subsystem may include a plurality of drivers for instantiating the archive on a variety of media. As new media becomes available, a driver for providing read and write access to that media may be added to the instantiation subsystem.

20 The system may conform to a client-server model in which the archive is maintained on a server, and the server responds to requests from a client which are transmitted to the server over a network. Depending on the request, the ingestion subsystem or instantiation subsystem may be invoked.

The system may also include a migration subsystem for migrating the archive
25 to a new medium; and a presentation subsystem for presenting one or more data

objects from the archive using a self-describing, infrastructure-independent presentation mechanism retrieved from the archive.

As with the instantiation subsystem, the migration subsystem may include a plurality of drivers for instantiating the archive on a variety of media. As new media
5 becomes available, a driver for providing read and write access to that media may be added to the migration subsystem. Moreover, the presentation subsystem may be configured to present data objects as retrieved from a query-able mechanism, or it may be configured to present data objects as retrieved from the archive.

In a second embodiment of the invention, a knowledge-based persistent
10 archive of a collection of data objects tangibly embodied on a processor readable medium is provided. The knowledge-based persistent archive comprises a self-describing, infrastructure-independent representation of a logical structure for the collection, a self-describing, infrastructure-independent representation of the data objects, and a self-describing, infrastructure-independent representation of knowledge
15 relevant to the collection. Optionally, the archive may also include a self-describing, infrastructure-independent representation of a presentation mechanism for presenting one or more data objects from the collection.

The first two elements of the archive—the self-describing, infrastructure-independent representation of a logical structure for the collection and a self-
20 describing, infrastructure-independent representation of the data objects in the collection—are as described in the previous embodiment.

The third element—the self-describing, infrastructure-independent representation of knowledge relevant to the collection—represents knowledge not embodied in the first two elements which is necessary or desirable for the purpose of

understanding the collection, and which may be included in the archive for the purpose of enhancing or contributing to its persistent quality.

The knowledge may be in the form of relationships between concepts relevant to the collection. The relationships may be logical or semantic relationships, such as mappings between concepts and attributes or elements of data objects. The relationships may also be temporal or procedural relationships, such as timing relationships that may exist between data objects in the collection. The relationships may also be spatial or structural relationships, and embody rules or constraints between certain elements or attributes of data objects. The relationships may also be algorithmic or functional relationships, such as algorithmic relationships identifying features within data objects. The relationships may be used to validate the collection during ingestion, instantiation, migration, or presentation processes.

In one example, the relationships may be expressed in a language such as Prolog. Prolog is a non-proprietary and infrastructure-independent language which is emerging as a standard. Thus, it as well qualifies as a self-describing, infrastructure-independent mode of expressing the relationships.

During ingestion of data objects into a knowledge-based persistent archive, the knowledge base of the archive may be used to verify the transformation of data objects into a self-describing, infrastructure independent form. Similarly, during instantiation of a knowledge-based persistent archive, the knowledge base may be used to verify data objects retrieved from the archive.

The knowledge base of the archive may also be used to validate the collection of data objects contained in the archive. In particular, it may be used to check the internal consistency of the archive, i.e., determine that it is consistent with several known rules and any noted exceptions to the rules.

The knowledge base of a persistent archive may also comprise a self-describing, infrastructure-independent, or executable representation of a transformation procedure. Various methods are possible which utilize such a transformation procedure.

5 First, a method of transforming data objects into a form capable of ingestion into the archive is possible which, in one embodiment, comprises the steps of retrieving the representation of the procedure from the archive, and executing the procedure to transform the data objects into a form ready for ingestion into the archive.

10 Second, a method of transforming data objects into a form capable of instantiation onto a query-able mechanism is possible which, in one embodiment, comprises the steps of retrieving the representation of the transformation procedure from the archive, retrieving from the archive one or more data objects in a self-describing, infrastructure independent form, and executing the procedure to transform
15 the data objects in the self-describing, infrastructure independent form into a form capable of being instantiated onto a query-able mechanism.

20 Third, a method of transforming data objects into occurrences of attribute or element values is also possible which comprises, in one embodiment, the steps of retrieving the representation of the transformation procedure from the archive, retrieving from the archive one or more data objects in a self-describing, infrastructure independent form, and executing the procedure to transform the data objects in the self-describing, infrastructure independent form into the occurrences of the attribute or element values.

The occurrences of attribute or element values may also be formed using data records tagged with attribute or element names. Moreover, inverted attribute indices may be formed from the occurrences.

These occurrences and/or inverted attribute indices may be used for a variety of purposes, including (1) validating the collection, (2) identifying knowledge to be added to the knowledge base of a knowledge-based persistent archive formed from the tagged data records, such as exceptional conditions, (3) confirming closure of attribute or element selection for a collection formed from the tagged data records, (4) obtaining useful information about a collection formed from the tagged data records, such as the degree of redundancy in the collection, (5) determining transformation procedures for a collection formed from the tagged data records, and (6) checking the internal consistency of a collection formed or to be formed from the tagged data records.

Furthermore, these occurrences and/or inverted attribute indices may be (1) transformed into tagged data records, (2) transformed into a form capable of being ingested into a persistent archive, (3) transformed into a form capable of being instantiated onto a query-able mechanism, or (4) transformed into a form capable of being presented to a user.

In a third embodiment of the invention, a knowledge-based persistent archive of a collection of data objects is provided which includes at least one self-describing, infrastructure-independent or executable specification. The specification may be used to validate the collection or put it in a form suitable for instantiation, presentation, migration, ingestion, etc. The archive may be tangibly embodied on a processor readable medium.

The archive comprises at least one representation of the collection or of the data objects; at least one self-describing, infrastructure-independent or executable specification of one or more transformations relating to the collection; and at least one self-describing, infrastructure-independent or executable specification of one or more rules encoding knowledge relevant to the collection.

In this embodiment, at least one of the representations of the collection may be (1) a self-describing, infrastructure-independent representation, (2) raw data, (3) data in a form capable of presentation, (4) data in a form capable of instantiation onto a query-able mechanism, (5) occurrences of attribute or element values, (6) one or more inverted attribute indices, (7) a topic map, or (8) data in a form capable of migration onto another medium. Furthermore, at least one of the representations of the collection may be a product of one of the transformations, or an input to one of the transformations.

Moreover, at least one of the transformations in this embodiment may be (1) content-preserving, (2) invertible, (3) configured to produce data objects in a form suitable for ingestion into the archive, (4) configured to produce data objects in a form suitable for instantiation onto a query-able mechanism, (5) configured to produce data objects in a form suitable for presentation, (6) configured to produce data objects in a form suitable for migration, (7) configured to produce occurrences of attribute or element values, or (8) configured to produce one or more inverted attribute indices.

A method of automatically placing one or more data objects from an archived collection into a form suitable for instantiation onto a query-able mechanism is also provided. In one embodiment, this method comprises the steps of retrieving from the archive a self-describing, infrastructure-independent or executable specification of one or more transformations relevant to the collection; retrieving from the archive a representation of one or more data objects in the collection; and executing the

specification to automatically place the one or more data objects into a form suitable for instantiation onto the query-able mechanism.

Also provided is a method of automatically validating a collection of data objects within a persistent archive. In one embodiment, this method comprises the steps of retrieving from the archive a self-describing, infrastructure-independent or executable specification of one or more rules relevant to the collection; and executing the specification to automatically validate the collection.

In the foregoing method, the step of validating the collection may be performed by producing occurrences of attribute or element values; and determining that the occurrences are consistent with the rules encoded by the specification and any valid exceptions.

A method of automatically presenting one or more data objects from a persistent archive of a collection of data objects is also provided. In one embodiment, this method comprises retrieving from the archive a self-describing, infrastructure-independent or executable specification of one or more transformations relevant to the collection; retrieving from the archive a representation of one or more data objects in the collection; and executing the specification to automatically place the one or more data objects from the collection in a form suitable for presentation.

A method of automatically placing an archived collection of data objects into a form suitable for migration to a new medium is also provided. In one embodiment, this method comprises retrieving from the archive a self-describing, infrastructure-independent or executable specification of one or more transformations relevant to the collection; and executing the specification to automatically place the collection into a form suitable for migration to a new medium.

Any of the foregoing methods may be tangibly embodied on a processor readable medium.

A system is also provided which includes an engine for executing self-describing, infrastructure-independent, or executable specifications. In one
5 embodiment, this system may further include a validation subsystem for validating the collection by commanding the engine to execute at least one self-describing, infrastructure-independent or executable specification encoding one or more rules relevant to the collection.

This system may further include a transformation subsystem for transforming
10 one or more data objects in the collection by commanding the engine to execute at least one self-describing, infrastructure-independent or executable specification of one or more transformations relevant to the collection.

Other systems, methods, features and advantages of the invention will be or
will become apparent to one with skill in the art upon examination of the following
15 figures and detailed description. It is intended that all such additional systems, methods, features and advantages be included within this description, be within the scope of the invention, and be protected by the accompanying claims.

BRIEF DESCRIPTION OF THE FIGURES

The invention can be better understood with reference to the following figures. The components in the figures are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention. Moreover, in the figures, like reference numerals designate corresponding parts throughout the different views.

Figure 1 illustrates an embodiment of a persistent archive according to the invention.

Figure 2 illustrates an example of raw data records.

Figure 3A illustrates an example of an XML DTD.

Figure 3B illustrates an example of an XML encoded data object.

Figure 4 illustrates an example of an XML encoded data object which incorporates a DTD.

Figure 5 illustrates another example of an XML encoded data object.

Figure 6 illustrates an example of an XSL style sheet.

Figure 7 illustrates an example of an HTML document specifying a format for a web browser.

Figure 8 illustrates an example of a raw e-mail message manipulated as an object.

Figure 9 illustrates an example of a DTD for the e-mail messages.

Figure 10 illustrates SQL commands for creating relational database tables for the e-mails.

Figure 11 illustrates an example of a presentation using Microsoft Notepad, an XML viewer.

Figure 12 illustrates an example of a web-based presentation formatted according to an HTML style sheet.

5 Figure 13 illustrates an example of a web-based presentation of the results of a database query formatted using an HTML style sheet.

Figure 14 illustrates an example of a DTD for Vietnam casualty statistics at the collection level.

10 Figure 15 is an example of a DTD for Vietnam casualty statistics at the data object level.

Figure 16 is an example of a SQL command for creating a relational database table for Vietnam casualty statistics.

Figure 17 is an example of a SQL query to detect incomplete records in a database of Vietnam casualty statistics.

15 Figure 18 is an example of an SQL query to determine total count, minimum age, maximum age, and average age at time of death from a database of Vietnam casualty statistics.

Figure 19 is a flowchart of an embodiment of a method of ingestion according to the invention.

20 Figure 20 is a flowchart of an embodiment of a method of instantiation according to the invention.

Figure 21A–21B are flowcharts of embodiments of methods of presentation according to the invention.

Figure 22 is a flowchart of an embodiment of a method of migration according to the invention.

5 Figure 23A illustrates one embodiment of a system according to the invention.

Figure 23B illustrates subsystems within the system of Figure 23A.

Figure 24 illustrates an embodiment of a knowledge-based persistent archive according to the invention.

Figure 25 illustrates an example of a DTD for legislative amendments.

10 Figure 26 illustrates an example of a DTD for legislative Acts.

Figure 27 illustrates an example of a declaration of an amendment element in the Act DTD.

Figures 28A-28C illustrate an example of an XML encoded Amendment.

15 Figure 29 gives the meanings of file prefixes for an example collection of legislative Acts and amendments.

Figure 30 gives the meanings of file suffixes for an example collection of legislative Acts and amendment.

Figures 31A-31R illustrate an example of a data dictionary for a collection of data objects representing artworks and images of artworks.

20 Figures 32A-32K illustrate an example of a DTD for a collection of data objects representing artworks and images of artworks.

Figure 33 illustrates the transformation of raw data records into XML encoded data objects.

Figure 34 illustrates an example of a presentation made according to an XSL style sheet.

5 Figures 35A-35G is a Perl script embodying a procedure for transforming raw data records into XML encoded data objects for an example collection involving artworks and images of artworks.

10 Figures 36A-36Z is a Perl script embodying a procedure for transforming XML encoded data objects into occurrence tuples for an example collection involving Senate legislative activities for the 106th Congress.

Figure 37 is an example of a DTD for an example collection involving Senate legislative activities for the 106th Congress.

Figure 38 is a Perl script for transforming raw data records into XML encoded data objects for an example collection involving Vietnam casualty statistics.

15 Figures 39A-39B is a Perl script for transforming raw data records into a form ready for instantiation onto a relational database for an example collection involving Vietnam casualty statistics.

Figure 40 is a flowchart of one embodiment of a method for ingesting data objects into a knowledge-based persistent archive according to the invention.

20 Figure 41A is a flowchart of one embodiment of a method of instantiating a knowledge-based persistent archive according to the invention.

Figure 41B is a flowchart of one embodiment of a method of presenting data objects from a knowledge-based persistent archive according to the invention.

Figure 42 is a flowchart of one embodiment of a method of validating a knowledge-based persistent archive according to the invention.

5 Figure 43 is a flowchart depicting one embodiment of the process of achieving closure of attribute selection according to the invention.

Figure 44A is a flowchart of one embodiment of a method of transforming data objects according to the invention.

10 Figure 44B is a flowchart of one embodiment of a method of transforming data objects into a form ready for instantiation onto a query-able mechanism according to the invention.

Figure 44C is a flowchart of one embodiment of a method of transforming data objects into occurrence tuples according to the invention.

15 Figure 45 is a flowchart of one embodiment of a method of forming occurrence tuples from tagged data objects according to the invention.

Figure 46 illustrates one embodiment of a knowledge-based persistent archive including at least one executable specification according to the invention.

Figure 47 illustrates one embodiment of an ingestion network according to the invention.

20 Figure 48 illustrates examples of F-logic rules implied by DTD declarations.

Figure 49 illustrates an example of an ingestion network for an example collection involving Senate legislative activities for the 106th Congress.

Figure 50 is a flowchart illustrating one embodiment of a method of transforming data objects into a form suitable for instantiation onto a query-able mechanism according to the invention.

5 Figure 51A is a flowchart illustrating one embodiment of a method of validating a collection according to the invention.

Figure 51B is a flowchart illustrating one example implementation of the method of Figure 51A.

10 Figure 52 is a flowchart illustrating one embodiment of a method of transforming data objects into a form suitable for presentation according to the invention.

Figure 53 is a flowchart illustrating one embodiment of a method of transforming data objects into a form suitable for migration to a new medium according to the invention.

15 Figures 54A-54C illustrate an example of a topic map for a collection involving Senate legislative activities for the 106th Congress.

DETAILED DESCRIPTION

I. First Embodiment

A. Persistent Archives

20 A first embodiment of the invention comprises a persistent archive as illustrated in Figure 1. The archive, which may be tangibly embodied on a processor readable medium, comprises a self-describing, infrastructure-independent representation of a logical structure for the collection, identified with numeral 100, and a self-describing, infrastructure-independent representation of the data objects, identified with numeral 102. As illustrated, the archive may also include a self-

describing, infrastructure-independent representation of a presentation mechanism for one or more of the data objects. This representation is identified in the figure with numeral 104.

For purposes of this disclosure, the phrase “self-describing” is a flexible concept which varies according to the circumstances, but it is generally used to refer to an element whose meaning is apparent from the element itself or through resort to no more than generally understood principles; the term “persistent” generally means the quality or capability of being accessible and usable at an indefinite point in time in the future; and the phrase “infrastructure-independent” generally refers to the state or quality of being independent of a particular storage or computing platform or implementation or at most limited to only a generic class of storage or computing platforms or implementations.

Since the elements of the persistent archive—the logical structure of the collection and the data objects—are expressed in a self-describing, infrastructure-independent form, the collection can be re-instantiated and understood at an indefinite point in time in the future no matter what the specific state of technology is at the time. Consequently, the archive is persistent.

The logical structure of the collection may be expressed through a variety of means, but, in one example, the logical structure is expressed in the form of an eXtensible Markup Language (XML) Document Type Definition (DTD), which defines elements of data objects or collections, their interrelationship, and their attributes. Since an XML DTD is a non-proprietary and widely known mode of expression, is platform-independent, and is emerging as a standard, it qualifies as a self-describing infrastructure-independent means of expressing the logical structure of the collection. In fact, a standards body, the World Wide Web Consortium (W3C), originated XML and continues to develop a specification for it today. XML 1.0

became a formal W3C Recommendation in February, 1998. The formal specification for XML 1.0 is available at www.w3.org/TR/REC-xml, and is reproduced as Appendix A of "Professional XML," Wrox Press, 2000, pp. 937-980, which is hereby fully incorporated by reference herein as through set forth in full. An annotated version of the specification is available at www.xml.com/axml/testaxml.htm.

An example of an XML DTD is illustrated in Figure 3B. This figure illustrates a DTD created for the customer records illustrated in Figure 2. As can be seen in Figure 2, each record associates the following fields with a customer: Customer Id, First, Last, City, Email, Phone, and Fax. This association is reflected in the DTD of Figure 3A. There, the portion identified with numeral 300 creates a root element, CUSTOMER, and associates it with the following child elements: customer_name, email, phone, and fax. The + symbol following the customer_name element indicates that element can appear one or more times for each CUSTOMER; the * symbol following the email and phone elements indicates that these elements are optional and, if present, can appear more than once; and the ? symbol following the fax element indicates that this element is optional and, if present, can only appear once.

In addition, the DTD, in the portion identified with numeral 302, associates the following attributes with the root element CUSTOMER: customer_ID and city. The ID nomenclature appearing next to the customer_id element indicates that element is a unique identifier of the CUSTOMER element. The customer_id and city attributes are further identified as being required for each CUSTOMER element.

Line 304 indicates that the customer_name element itself has two child elements, first and last, each of which can appear one or more times. Portion 306 indicates that the first, last, email, phone, and fax elements may each have content which comprises PCDATA, i.e., parsed character data.

In another example, the logical structure of the collection is expressed in the form of Structured Query Language (SQL) commands for creating relational database tables. Since SQL is a non-proprietary and widely known mode of expression, and is platform independent, this mode of expression also qualifies as self-describing. This form of expression is described farther on in this specification in relation to Figure 10.

The data objects of the collection may also be expressed through a variety of means, but, in one example, the data objects are expressed in the form of tagged XML data objects, in which components of the data objects are tagged with element or attribute names from the DTD. Since the DTD defines the meaning and interrelationship of the elements and attributes, the tagging, in associating element and attribute names with components of the data objects, qualifies as a self-describing, infrastructure-independent form of expression.

Figure 3B is an example of a tagged XML document, and is the first of the customer records of Figure 2 tagged using the DTD of Figure 3A. Portion 308 indicates that the record is tagged using XML 1.0, and also identifies the corresponding DTD. Portion 310 consists of the tags identifying the root element CUSTOMER, and the values of its two attributes, customer_id and city. Portion 312 consists of the tags for the child element customer_name, and portion 314 shows the parsed character strings tagged by the child elements first and last. Finally, portion 316 shows the parsed character strings tagged by the elements email, phone, and fax.

Note that XML allows the integration of a DTD into a tagged XML document. Figure 4 illustrates an example of such a document. Portion 400 indicates that the DTD is integrated into the XML document, portion 402 is the DTD, and portion 404 consists of the tagged XML document.

The persistent archive may also include a self-describing, infrastructure-independent representation of a presentation mechanism for one or more of the data objects. The presentation mechanism may capture the “look and feel” of certain presentation formats for the data objects that may be re-created at some point in the future when the archive is re-instantiated.

In one example, the presentation mechanism is represented in the form of an eXtensible Stylesheet Language (XSL) style sheet which specifies one or more templates for transforming XML-tagged data objects into desired presentation entities, such as a HTML page for presentation on a web browser. Since XSL is written in XML, it as well qualifies as a self-describing, infrastructure-independent form of expression. In December 1998, the standards body W3C made a formal Recommendation of the transformations portion of XSL (known as XSLT). An overview of XSL is available in “Professional XML,” *supra*, Appendix E, pp. 1085-1099, and pp. 373-418, 692-712, which sections are all hereby fully incorporated by reference herein as though set forth in full.

Figure 6 is an example of an XSL style sheet for transforming the tagged XML document of Figure 5 into the HTML page of Figure 7. The HTML page represents a mechanism for presenting the tagged XML document on a web browser.

A first example of a persistent archive according to the invention will now be described. This first example is a persistent archive of a collection of e-mails, one of which is illustrated in Figure 8. The persistent archive in this example consists of the combination of (1) a DTD specifying elements and attributes for the e-mails, illustrated in Figure 9; (2) SQL commands for creating relational database tables, illustrated in Figure 10; (3) the XML-tagged e-mails; and (4) an HTML web page for presenting e-mails on a web browser.

Note that the DTD of Figure 9 creates elements for most of the fields in the e-mail of Figure 8, and divides these elements into three groupings: required, optional, and other. Note also that the SQL commands of Figure 10 and the DTD of Figure 9 both represent a logical structure of the collection, although for different purposes.

5 The SQL commands of Figure 10 represent a logical structure for the collection which is appropriate for the purpose of instantiating the archive as a relational database. The DTD of Figure 9, on the other hand, depicts the structure of individual data objects within the collection for the purpose of validating tagged XML documents using an XML parser. Although Figure 9 depicts the structure of an individual data object, it
10 still represents the structure of the entire collection, since the collection is an accumulation of individual data objects.

Figure 11 illustrates a presentation of the e-mails using Microsoft Notepad, an XML viewer. Note that the viewer displays the tagged XML documents directly, and does not utilize the HTML page for this purpose. Figure 12 illustrates a query to the
15 e-mails after they have been instantiated onto a relational database. Figure 13 illustrates the results of the query displayed according to the HTML page stored with the archive. As can be seen, the results are stored in blocks of 10, and the text of the e-mail currently clicked on is displayed in a separate tinted window.

A second example of a persistent archive in accordance with the invention will
20 now be described. This second example concerns Vietnam casualty statistics. Here, the collection comprises a plurality of data objects, each of which is a record documenting a Vietnam casualty. The persistent archive in this example comprises (a) a DTD for the collection as a whole, illustrated in Figure 14; (b) a DTD for the individual data objects within the collection, illustrated in Figure 15; (c) SQL
25 commands for creating relational database tables for the collection, illustrated in Figure 16; and (d) XML-tagged casualty records.

Note that the DTD of Figure 14, the DTD of Figure 15, and the SQL commands of Figure 16, each represent the logical structure of the collection, although for different purposes. The DTD of Figure 14 is for the purpose of defining elements at the collection level, and the DTD of Figure 15 is for the purpose of defining elements at the individual data object level. The tables created by the SQL commands of Figure 16 are for the purpose of instantiating the collection as a relational database. Note also that the XML-tagged records represent the data objects of the collection.

Referring to Figure 14, the DTD indicates that the collection has one or more databases, and the remainder of the DTD indicates that each database has a creation date, a record size, and one or more records.

Referring to Figure 15, the DTD indicates that a record has no child elements, but instead is defined in terms of required attributes, such as social security or service number, date of death, date of birth, cause of death, age at time of death, etc.

Referring to Figure 16, the SQL commands create a relational database table with entries for each of the required attributes in the DTD of Figure 15. Other records which are added include a unique record number and a record indicating that the social security number/service number is the primary key for querying the database.

Figures 17 and 18 illustrate examples of queries which can be made once the collection has been instantiated as a relational database. Figure 17 illustrates a SQL query seeking all casualty records where the date of birth or date of death is incomplete or null. Figure 18 illustrates a SQL query seeking a total count of the casualty records where age at time of death is available, the minimum age and maximum age represented by all these records, and the average age determined over all these records.

B. Methods Involving Persistent Archives

The invention includes several methods involving persistent archives such as a method of ingesting data objects into a persistent archive, a method of instantiating a persistent archive onto a query-able mechanism, a method of presenting one or more data objects from a persistent archive, and a method of migrating a persistent archive from a first medium to a second medium.

Figure 19 is a flowchart of one embodiment of a method of ingesting one or more data objects into a persistent archive according to the invention. This method may be practiced when the archive is first created, or when an existing archive is to be supplemented. As illustrated, the method comprises steps 1900 and 1902. Step 1900 comprises transforming a representation of the data objects into a self-describing, infrastructure-independent representation of the data objects. In one example, this step comprises tagging the content of the data objects with XML element and attribute names, thus giving rise to XML-tagged data objects. Step 1902 comprises archiving the self-describing, infrastructure-independent representation of the data objects with a self-describing, infrastructure-independent representation of a logical structure for the collection.

In one example, the self-describing, infrastructure-independent representation of a logical structure for the collection is a DTD for the individual data objects in the collection. In a second example, it is a DTD for the overall collection. In a third example, it is a representation of the collection suitable for instantiation on a query-able mechanism. This representation may comprise SQL commands for creating one or more relational database tables embodying a relational database schema. In a fourth example, this representation may comprise combinations of the foregoing.

Figure 20 is a flowchart of a method of instantiating a persistent archive onto a query-able mechanism according to one embodiment of the invention. This method

may be practiced when it is desired to re-establish the archive onto a query-able mechanism which is able to access and retrieve the data objects in response to specific queries. That may occur many years after the archive is first created, and may even involve instantiating the archive onto technology which was not in existence at the time the archive was created.

As illustrated, the method comprises, in step 2000, retrieving from the persistent archive a self-describing, infrastructure-independent representation of a logical structure for the collection; in step 2002, creating on a medium a query-able mechanism in accordance with the logical structure; in step 2004, retrieving from the archive a self-describing, infrastructure-independent representation of one or more data objects; and then, in step 2006, loading the one or more data objects into the query-mechanism.

In one example, the query-able mechanism is a database management system, such as a relational or hierarchical database management system. The data objects, once instantiated on the database management system, may be rapidly accessed using database queries. The retrieved objects may then be presented using a presentation mechanism retrieved from the archive. In one example, the presentation mechanism is an HTML web page which specifies the format for displaying data objects on a web browser. In another example, the presentation mechanism is a XSL style sheet which specifies a format for displaying XML-tagged data objects on a web browser.

Figure 21A is a flowchart of a method of presenting, according to one embodiment of the invention, one or more data objects from a query-able mechanism onto which a persistent archive has been instantiated. The method comprises, in step 2100, retrieving from the archive a self-describing, infrastructure-independent representation of a presentation mechanism for one or more data objects in the archive; in step 2102, querying the query-able mechanism for one or more data

objects, in step 2104, retrieving from the query-able mechanism the one or more data objects queried in the previous step; and, in step 2106, presenting the one or more data objects using the presentation mechanism.

Figure 21B is a flowchart of a method, according to one embodiment of the invention, of presenting one or more data objects retrieved from a persistent archive. The method comprises, in step 2108, retrieving from the archive a self-describing, infrastructure-independent representation of a presentation mechanism for one or more data objects; in step 2110, retrieving from the archive a self-describing, infrastructure-independent representation of one or more of the data objects; and, in step 2112, presenting the one or more data objects using the presentation mechanism.

Figure 22 is a flowchart of a method, according to one embodiment of the invention, of migrating a persistent archive from a first medium onto a second medium. For purposes of this disclosure, a “medium” is any processor accessible device capable of storing data, including without limitation RAM, ROM, EPROM, EEPROM, PROM, disk, floppy disk, hard disk, CD-ROM, DVD, flash memory, etc. This method may be practiced on a periodic basis to guard against physical degradation or obsolescence of the medium on which a persistent archive is stored. The second medium may even embody new technology which was not in existence at the time the archive was stored on the first medium.

The method comprises, in step 2200, retrieving the persistent archive from a first medium; in step 2202, optionally redefining the logical structure of the collection or the self-describing, infrastructure-independent representation of the data objects in the archive; and, in step 2204, storing the persistent archive as optionally redefined onto a second medium.

Each of the foregoing methods may be tangibly embodied as a series of processor-executable instructions stored on a processor-readable medium. Again, for purposes of this disclosure, a “medium” is any processor accessible device capable of storing data, including without limitation RAM, ROM, EPROM, EEPROM, PROM, disk, floppy disk, hard disk, CD-ROM, DVD, flash memory, etc.

In one example, one or more of the foregoing methods are tangibly embodied as a Perl script stored on a processor readable medium. Examples of Perl scripts for performing various ingestion and instantiation functions are illustrated in Figures 32A-32K, 35A-35G, 36A-36Z, 38, and 39A-39B, which are discussed infra.

C. Systems For Maintaining Persistent Archives

A system for maintaining a persistent archive according to one embodiment of the invention is illustrated in Figure 23A. As illustrated, in this embodiment, the system conforms to a client-server model, but it should be appreciated that embodiments are possible which conform to other models, such as a typical processor configuration in which a processor is tightly coupled to one or more storage devices or media, and one or more input/output (I/O) devices through an address and data bus, and also possible an I/O bus.

Referring to Figure 23A, the system comprises one or more clients 2300a, 2300b, 2300c capable of accessing one or more servers 2304a, 2304b, 2304c over one or more networks 2302. The network 2302 may be a proprietary network or a public network such as the Internet. Moreover, the clients may be coupled to the network through wireline or wireless communications links 2310, and the servers may be coupled to the network through wireline or wireless communications links 2312.

One or more of the servers 2304a, 2304b, 2304c may include a medium 2306a, 2306b, 2306c capable of holding a persistent archive or a query-able mechanism onto

which a persistent archive may be instantiated. One or more of the clients 2300a, 2300b, 2300c may be capable of holding a presentation mechanism for presenting data objects from one of the archives or query-able mechanisms maintained on the servers.

5 One or more the clients 2300a, 2300b, 2300c may be capable of issuing requests which are provided to one or more of the servers 2304a, 2304b, 2304c over the network 2302. Responsive thereto, one or more servers receiving the requests are configured to respond to and handle the requests appropriately.

10 Referring to Figure 23B, one or more of the servers 2304a, 2304b, 2304c may comprise an ingestion subsystem 2310 for ingesting data objects into a persistent archive, responsive to a request from a client, and an instantiation subsystem 2312 for instantiating a persistent archive onto a query-able mechanism, again responsive to a request from a client. The instantiation subsystem 2312 may include a plurality of drivers 2314a, 2314b, 2314c for instantiating the archive on a variety of media. As
15 new media becomes available, a driver for providing read and write access to that media may be added to the instantiation subsystem 2312.

20 Again referring to Figure 23B, one or more of the servers may also include a migration subsystem 2318 for migrating an archive onto a new medium, perhaps maintained by a different server, responsive to a request from a client, and a presentation subsystem 2316 for presenting to a client, responsive to a request from a client, one or more data objects from an archive maintained either on the server or a query-able mechanism maintained by the server onto which the archive has been instantiated. The one or more data objects may be presented to the client, and the client may in turn present the one or more data objects to a user, using a self-
25 describing, infrastructure-independent presentation mechanism retrieved from the archive.

As with the instantiation subsystem 2312, the migration 2318 subsystem may include a plurality of drivers 2320a, 2320b, 2320c for instantiating the archive on a variety of media. As new media becomes available, a driver for providing read and write access to that media may be added to the migration subsystem 2318.

5 As discussed, the system may conform to a client-server model in which an archive is maintained on a server, and the server responds to requests from a client which are transmitted to the server over a network. Depending on the request, the ingestion subsystem 2310, instantiation subsystem 2312, presentation subsystem 2316, or migration subsystem 2318 may be invoked. Requests for ingestion are
10 handled by the ingestion subsystem 2310, requests for instantiation are handled by the instantiation subsystem 2312, requests for presentation are handled by the presentation subsystem 2316, and requests for migration are handled by the migration subsystem 2318.

In one implementation, the system is a Storage Resource Broker as developed
15 and maintained at the Supercomputer Center on the campus of the University of California, San Diego. In this implementation, the SRB is middleware which brokers requests from clients in a client-server system to servers. When a request by a client is initiated, the SRB spawns a broker to handle the request. The broker then examines system resources and selects one or more servers to handle the request. The broker
20 then passes the request on to the selected servers which then handle the request. The broker monitors the situation and remains active until the request has been handled. At that point, the broker reports any results back to the client, and also updates it on the status of the request, i.e., whether it was satisfactorily completed or not.

The SRB may also employ a meta-data catalog (MCAT) for keeping track of
25 where data is stored throughout the system. When a request from a client necessitates access to data stored on the system, the broker which is spawned to handle the request

may first access the MCAT to determine where the data is located on the system. If the request results in the data being altered or modified in any way, the broker may update the MCAT to reflect these modifications and changes. Similarly, if the request results in the addition of data to the system, the broker may create records in the MCAT indicating the location of, and possibly other attributes, of this data.

II. Second Embodiment

A. Knowledge-Based Persistent Archives

In a second embodiment, the invention provides a knowledge-based persistent archive of a collection of data objects tangibly embodied on a processor readable medium. Referring to Figure 24, one embodiment of the knowledge-based persistent archive comprises a self-describing, infrastructure-independent representation of a logical structure for the collection, identified with numeral 2400, a self-describing, infrastructure-independent representation of the data objects, identified with numeral 2402, and a self-describing, infrastructure-independent representation of knowledge relevant to the collection, identified with numeral 2404. The archive may optionally include a self-describing, infrastructure-independent representation of a presentation mechanism for one or more data objects in the collection.

The first two elements of the archive—the self-describing, infrastructure-independent representation of a logical structure for the collection and a self-describing, infrastructure-independent representation of the data objects in the collection—are as described in relation to the previous embodiment. Similarly, the fourth optional element—the self-describing, infrastructure-independent representation of the data objects in the collection—is as described in relation to the previous embodiment.

The third element—the self-describing, infrastructure-independent representation of knowledge relevant to the collection—represents knowledge which may not be embodied in the first two elements and which may be necessary or desirable for the purpose of understanding the collection. Accordingly, this element
5 may be included in the archive for the purpose of enhancing or contributing to its persistent quality.

The knowledge may be in the form of relationships between concepts relevant to the collection. The relationships may be logical or semantic relationships, such as mappings between concepts and attributes or elements of data objects. The
10 relationships may also be temporal or procedural relationships, such as timing relationships that may exist between data objects in the collection, or procedures for transforming data objects into a form ready for ingestion into the archive, instantiation into a query-able mechanism, or presentation to a user. The relationships may also be spatial or structural relationships, and embody rules or constraints between certain
15 elements or attributes of data objects. The relationships may also be algorithmic or functional relationships, such as algorithmic relationships between data objects and features of data objects.

In one example, the knowledge may be expressed in a language such as Prolog. Prolog is a non-proprietary and infrastructure-independent language which is
20 emerging as a standard. Thus, it as well qualifies as a self-describing, infrastructure-independent mode of expressing the knowledge. In a second example, the knowledge may be expressed in the form of Knowledge Interchange Format (KIF). In a third example, the knowledge is in the form of a topic map which maps concepts relevant to the collection to attribute and elements of the data objects in accordance with the
25 ISO 13250 Topic Map standard (set forth in ISO/IEC 13250 Topic Maps, International Organization for Standardization and the International

Electromechanical Commission, which is hereby fully incorporated herein as though set forth in full).

A first example of a knowledge-based persistent archive according to the invention will now be described. This archive concerns a collection of data objects each representing a particular legislative action, i.e., bill, amendment, resolution, etc., taken during the 104th Congress. The raw data for ingestion into the archive consists of a plurality of files, each representing a legislative action, with a prefix to the file name indicating the origin of the legislative action, and a suffix to the file indicating the specific type of legislative action represented by the file.

Referring to Figure 29, the possible prefixes to the files are indicated. Generally speaking, the prefixes identify whether the legislative action originated in the House or the Senate. Figure 30 lists the possible suffixes to the files. For example, the suffixes “eah” and “eas” both indicate that the corresponding legislative action is an amendment, and the suffixes “eh” and “es” both indicate that the corresponding legislative action is an Act.

The ingestion process in this particular example consists of creating a separate DTD for each possible type of legislative action, and then tagging each of the files with the elements and attributes of the corresponding DTD to create XML encoded data objects, each representing a legislative action.

In this particular example, the knowledge base that is considered important is the timing relationships between the various legislative actions represented by the suffixes in Figure 30. This is considered important because these timing relationships embody relationships that exist between the various data objects in the collection. For example, consider that an enacted piece of legislation starts out as a bill in one of the bodies of Congress (House or Senate). That bill is eventually referred to a

Committee. After emerging from the Committee, the bill is eventually referred to the full House or Senate, where it may be read on the floor of these bodies. In addition, that bill spawns a bill in the other body of Congress. That bill in turn is also referred to a Committee, and eventually to the full House or Senate. A joint Committee then
5 resolves differences between the House and Senate versions of the bill. The data objects representing these legislative actions all bear a relationship to one another, and the purpose of the knowledge base in this particular example is to capture these relationships. In one implementation, this information is captured and expressed in a language such as Prolog.

10 The persistent archive in this example comprises (a) the DTD for each of the possible legislative actions; (b) the XML-tagged data objects each representing a legislative action taken during the 104th Congress; and (c) the timing relationships between the various legislative actions expressed in Prolog.

15 Figure 25 illustrates an example of a DTD for amendments, i.e., those files having a suffix of “eas” or “eah”. The core portion of this DTD is the element RESOLUTION. As indicated, the element RESOLUTION consists of (1) zero or more instances of the element TYPE or (2) the element STATEMENT.

Moreover, the element STATEMENT consists of (1) the element OPENING
20 or (2) zero or more instances of the element STRIKEOUT or the element INSERT or (3) the element CLOSING.

Figures 28A-28C illustrate an example of an XML tagged data object for an amendment. In this particular example, there is an OPENING, two STRIKEOUTS, two INSERTS, and a CLOSING.

25 Figure 26 illustrates one example of the DTD for an Act, i.e., those files having a suffix of “eh” or “es”. The core portion of this DTD is the element ACT.

As indicated, this element consists of zero or more instances of the elements PURPOSE or SECTION. The element SECTION consists of zero or more instances of (1) HEADING or (2) STATEMENT or (3) SUBSECTION. The element HEADING consists of zero or more instances of (1) character data or (2) the element SHORTTITLE.

The element STATEMENT consists of zero or more instances of (1) character data or (2) the element AMENDMENT or (3) the element SECTION or (4) the element ATTESTATION.

The element AMENDMENT consists of zero or more instances of (1) the element CODE or (2) the element STRIKEOUT or (3) the element INSERT or (4) the element REDESIGNATE.

The element INSERT consists of zero or more instances of (1) character data or (2) the element PARAGRAPH or (3) the element SUBSECTION or (4) the element INSERT.

The element PARAGRAPH consists of zero or more instances of (1) the element TOPIC or (2) the element TEXT or (3) the element PARAGRAPH. An attribute NUMBER is associated with each PARAGRAPH.

The element SUBSECTION consists of zero or more instances of (1) the element TOPIC or (2) the element PARAGRAPH. An attribute NUMBER is associated with each SUBSECTION.

Note that, in this example, an Act can be an original Act or an amendment. Thus, the DTD of Figure 26 can also be used in lieu of the DTD of Figure 25 to tag files representing amendments. Figure 27 illustrates the portion of the DTD of Figure

26 which gives it the flexibility to handle amendments: the declaration that the element STATEMENT can consist of the element AMENDMENT.

A second example of a knowledge-based persistent archive according to the invention will now be described. This example involves a collection of data objects
5 each of which represents an art object as well as one or more images of the art object. Accordingly, the DTD in this example specifies elements and attributes of both the underlying art object as well as any images of the art object.

This combined DTD is illustrated in Figures 32A-32K. As indicated, for each art object, the DTD creates elements for classifying the work (am_classification),
10 describing its appearance (am_appearance), identifying its creators (am_creators), etc. In addition, the DTD creates an element (am_media_metadata) which contains links to images and multimedia files relating to the art object and to data which further describes these images and multimedia files.

This DTD is in accordance with a data dictionary supplied by the producer of
15 the archive. This data dictionary, which is expressed in the form of a Microsoft Excel spreadsheet, is illustrated in Figures 31A-31S. In particular, note that the DTD specifies atomic elements for the record fields described in the data dictionary. For example, Figure 31A of the data dictionary describes a OTY record type for identifying the kind of art work involved, and Figure 32B of the DTD specifies the
20 atomic element OTY_object_type as a child of the am_classification element.

This DTD can be used to tag raw data records for each of the art objects and their corresponding images/multimedia files. Figure 33 is a screen shot showing on the left an example of a raw data record, and also showing on the right the XML-tagged version of that data record.

An XSL style sheet can be used to display the XML tagged data objects on a web browser in a prescribed format. Figure 34 is a screen shot showing at the upper left an XSL style sheet, showing on the lower left XML-tagged data objects, and showing on the right a screen shot of the data objects presented in accordance with the XSL style sheet.

A Perl script can be used for the purpose of automatically transforming the raw data records into XML tagged data objects. This script is set forth at Figures 35A-35G. Note that Perl is a standard language capable of being executed on a variety of platforms. Thus, it qualifies as a self-describing, infrastructure-independent representation of the process of transforming the raw data records into the XML tagged data objects.

The knowledge base in this example consists of two items. The first is the data dictionary of Figures 31A-31S expressed in a suitable self-describing, infrastructure-independent form such as Prolog or KIF. This data dictionary is useful to archive because it contains a definition of each of the record fields, and thus the atomic elements in the DTD. Thus, it can be used to facilitate understanding of the archive at the time of instantiation.

The second element is the Perl script of Figures 35A-35G. This script is useful to archive because it can be used for the purpose of ingesting additional data records into the archive at some point in the future. When it is desired to supplement the archive with new data records, this script could be retrieved and used to transform the raw data records into XML tagged data objects which would then be added to the archive. In addition, since this script provides the processing steps used to create the data objects in the collection, it can facilitate understanding of the collection at some point in the future.

The knowledge-based persistent archive in this example thus comprises: (1) the DTD of Figures 32A-32K; (2) the XML tagged data objects representing the art objects and their associated images/multimedia files; (3) optionally, one or more XSL style sheets for presenting the data objects; (4) the data dictionary of Figures 31A-31S expressed in a language such as Prolog or KIF; and (5) the Perl script of Figures 35A-35G.

A third example of a knowledge-based persistent archive in accordance with the invention will now be described. This third example involves Senate legislative activities for the 106th Congress. Raw files each represent a legislative activity, whether a bill, resolution, or amendment, by a Senator during the 106th Congress. Each of the files is encoded into an XML-tagged format in accordance with the DTD of Figure 37. As can be seen, this DTD specifies a Senate legislative activity (SLA) collection consisting of zero or more senate_files. Each senate_file may have a header_page which identifies the Senator by first name, middle name, last name, and state. In addition, each senate_file may have zero or more sections, with each section associated with one or more bills, amendments, or resolutions (bars).

A Perl script, illustrated in Figures 36A-36Z, is capable of processing the XML tagged data objects and producing a list of occurrence, attribute, value (oav) triplets. Each oav triplet represents an occurrence of an attribute of the data object (element using XML terminology), and identifies (1) the attribute, (2) the value associated with that attribute for the particular occurrence, and (3) the location of that occurrence in the XML tagged data objects.

These oav triplets may be used for a variety of purposes. First, they may be used to confirm that the selected XML elements and attributes are suitable for representing either the collection or the data objects. For example, if an XML element is selected, but there are no oav triplets for the element, that suggests the

element may not be needed to represent the data objects, and that a new element may be appropriate. As another example, the oav triplets may help to determine that a combination of elements, e.g., first and last name, may not be sufficient to uniquely identify a Senator and that it may be necessary to add the state of the Senator to the combination in order to form a unique identifier for a Senator. This helps achieve closure faster during the element or attribute definition phase.

The attribute selection process is represented in Figure 43. In step 4300, the accession template, i.e., the fields of the raw data records to be ingested into the archive, is analyzed, and in step 4302, attribute and elements relevant to the collection are initially selected. Then, in step 4304, the raw data records are tagged using the attribute and elements identified in the previous step. Step 4306 is then performed. There, oav triplets as described previously are produced. Step 4308 is then performed. This step comprises inverting the oav triplets to result in the following format: attribute(value, occurrence). This form facilitates determination of whether closure has been achieved. Step 4310 is then performed. This step comprises analyzing the oav triplets and/or the inverted triplets to ensure both completeness and consistency. Completeness may be identified when all attributes are populated, and the information content is fully annotated. Consistency may be identified when all attribute values fall within defined ranges, and when other relevant consistency checks are satisfied.

Second, the triplets may form a flat database which may be queried to obtain useful information about the collection. For example, these queries can help determine the amount of redundancy in the collection, and thus suggest transformation rules for reducing the size of the collection. In one instance, an analysis of the occurrence for the bar_id attribute revealed that only about 5,600 unique bars are represented by 44,000 SLA data objects. As another example, the queries can help determine if there are any exceptional or surprising conditions, such

as no legislative activity for a particular Senator, which would be useful to add to the knowledge base of the archive.

Third, the oav triplets may help check the internal consistency of the collection. For example, a particular Senator may be either a sponsor, co-sponsor, or submitter of a bar. Thus, it would seem reasonable to conclude that the total number of SLA files would equal the number of Senators identified within the collection as being a sponsor, co-sponsor, or submitter of a BAR. The oav triplets can be analyzed to determine whether or not this condition is satisfied, or whether exceptions exist which must be added to the knowledge base so that completeness and/or consistency can be declared.

Fourth, the oav triplets may be readily transformed into other useful data structures. For example, the oav triplets may be readily converted into a form ready for instantiation onto a query-able mechanism, such as a relational database. As another example, the oav triplets may be readily converted into a form, such as XML tagged data objects, ready for ingestion into the archive.

In this example, the knowledge base may consist of the Perl script of Figures 36A-36Z, the oav triplets, or the oav triplets in combination with the Perl script. The knowledge-based persistent archive in this example may thus comprise (1) the DTD of Figure 37; (2) the XML tagged data objects representing the SLA activities; and (3) the Perl script of Figures 36A-36Z and/or the oav triplets.

A fourth example of a knowledge-based persistent archive is based on the Vietnam casualty archive discussed in relation to the first embodiment. Here, this archive is extended to include a knowledge base. The knowledge base in this example may have two components. The first is a Perl script for automatically transforming the raw casualty records into XML tagged data objects using the DTD of

Figure 15. This first Perl script is illustrated in Figure 38. The second is a Perl script for automatically transforming the raw casualty records into a form ready for instantiation into an Oracle relational database. This second script is illustrated in Figures 39A-39B.

B. Methods Relating to Knowledge-Based Persistent Archives

The methods described previously in Section I.B. apply equally to knowledge-based persistent archives. However, there are several refinements and/or additions to these methods that take advantage of or utilize the knowledge base of the archive for a particular purpose.

During ingestion of data objects into a knowledge-based persistent archive, the knowledge base of the archive may be used to verify the data objects after transformation into a self-describing, infrastructure independent form. A flowchart of one embodiment of such a method is illustrated in Figure 40. As illustrated, the method comprises, in step 4000, transforming a representation of data objects into a self-describing, infrastructure-independent representation, and, in step 4002, verifying the transformed data objects using knowledge relevant to the collection.

In one implementation, where the self-describing, infrastructure independent representation of the data objects are data objects tagged with attribute or element names, this verification step may occur by forming occurrences of attribute or element values, inverting the occurrences to form inverted attribute indices, and using the inverted attribute indices to confirm that attribute and element values are within defined ranges, and that internal consistency of the collection is maintained.

Step 4004 follows step 4002. In step 4004, the self-describing, infrastructure independent representation of the data objects is archived with the self-describing, infrastructure independent representation of a logical structure of the collection and a

self-describing, infrastructure independent representation of knowledge relevant to the collection.

Similarly, during instantiation of a knowledge-based persistent archive, the knowledge base may be used to verify data objects retrieved from the archive. A flowchart of one embodiment of such a method is illustrated in Figure 41A. As illustrated, the method comprises, in step 4100, retrieving from the archive a self-describing, infrastructure independent representation of a logical structure of the collection, and, in step 4102, retrieving from the archive a self-describing, infrastructure independent representation of knowledge relevant to the collection. Step 4102 is followed by step 4104, in which a query-able mechanism is created in accordance with the logical structure of the collection.

In step 4106, a self-describing, infrastructure independent representation of one or more data objects is retrieved from the archive. Then, in step 4108, the retrieved data objects are verified to ensure that they are consistent with the knowledge relevant to the collection.

In one implementation, the self-describing, infrastructure independent representation of the data objects comprises data objects tagged with attribute or element names. In this implementation, the verifying step may occur by forming occurrences of attribute or element values from the tagged data objects, forming one or more inverted attribute indices from the occurrences, and confirming that the attribute or element values are within defined ranges, and/or follow prescribed rules and/or exceptions to the rules, and that the collection is internally consistent.

Step 4108 is followed by step 4110, in which the verified data objects are loaded into the query-able mechanism.

Once the data objects have been loaded into the query-able mechanism, they may be retrieved in response to queries for presentation to a user. At that time, they may be again be verified using the knowledge relevant to the collection as described above.

5 A flowchart of one embodiment of such a method is illustrated in Figure 41B. As illustrated, this method comprises, in step 4112, retrieving from the archive a self-describing, infrastructure-independent representation of a presentation mechanism for one or more of the data objects, and, in step 4114, retrieving, responsive to queries, one or more data objects from a query-able mechanism.

10 Step 4114 is followed by step 4116, which comprises verifying that the one or more retrieved data objects are consistent with the knowledge relevant to the collection. Again, this step may occur by forming occurrences of attribute or element values from tagged data objects, forming inverted attribute indices from the occurrences, and analyzing the inverted attribute indices to ensure that the attribute or
15 element values are within prescribed ranges and/or that the attribute or element values follow prescribed rules and/or exceptions to the rules, and that the collection is internally consistent.

Step 4116 is followed by step 4118. There, the one or more verified data objects are presented using the presentation mechanism.

20 The knowledge base of the archive may also be used to validate the collection of data objects contained in the archive at arbitrary times. A flowchart of one embodiment of such a method is illustrated in Figure 42. As illustrated, the method comprises, in step 4200, retrieving from the archive a self-describing, infrastructure independent representation of knowledge relevant to the collection, and, in step 4202,
25 using the knowledge to validate the collection.

Again, this step may occur by forming occurrences of attribute or element values from tagged data objects, forming inverted attribute indices from the occurrences, and analyzing the inverted attribute indices to ensure that the attribute or element values are within prescribed ranges and/or that the attribute or element values follow prescribed rules and/or exceptions to the rules, and that the collection is internally consistent.

The knowledge base of a persistent archive may also comprise a self-describing, infrastructure independent, executable representation of a transformation procedure, such as a Perl script. Various methods are possible which utilize such a transformation procedure.

First, a method of transforming data objects into a form capable of ingestion into the archive is possible. One embodiment of such a method is illustrated in Figure 44A. As illustrated, this embodiment of the method comprises, in step 4400, retrieving the representation of the procedure from the archive, and, in step 4402, executing the procedure to transform the data objects into a form ready for ingestion into the archive.

Second, a method of transforming data objects into a form capable of instantiation onto a query-able mechanism is also possible. One embodiment of such a method is illustrated in Figure 44B. As illustrated, the embodiment of the method comprises, in step 4404, retrieving the representation of the transformation procedure from the archive, and, in step 4406, retrieving from the archive one or more data objects in a self-describing, infrastructure independent form. This is followed by step 4408, which comprises executing the procedure to transform the data objects in the self-describing, infrastructure independent form into a form capable of being instantiated onto a query-able mechanism.

Third, a method of transforming data objects into occurrences of attribute or element values is also possible. One embodiment of such a method is illustrated in Figure 44C. As illustrated, the embodiment of the method comprises, in step 4410, retrieving the representation of the transformation procedure from the archive, and, in
5 step 4412, retrieving from the archive one or more data objects in a self-describing, infrastructure independent form. This is followed by step 4414, which comprises executing the procedure to transform the data objects in the self-describing, infrastructure independent form into the occurrences of attribute or element values.

The occurrences of attribute or element values may also be formed using data
10 records tagged with attribute or element names. An embodiment of such a method is illustrated in Figure 45. As illustrated, this embodiment of the method comprises, in step 4500, receiving data records tagged with attribute or element names, and, in step 4502, forming from the tagged data records occurrences of attribute or element values. The embodiment of the method may also include forming inverted attribute
15 indices from the occurrences.

These occurrences and/or inverted attribute indices may be used for a variety of purposes, including (1) validating the collection, (2) identifying knowledge to be added to the knowledge base of a knowledge-based persistent archive formed from the tagged data records, such as exceptional conditions, (3) confirming closure of
20 attribute or element selection for a collection formed from the tagged data records, (4) obtaining useful information about a collection formed from the tagged data records, such as the degree of redundancy in the collection, (5) determining transformation procedures for a collection formed from the tagged data records, (6) checking the internal consistency of a collection formed or to be formed from the tagged data
25 records, and (7) confirming that the attribute or element values fall within prescribed ranges, and/or that the attribute or element values follow prescribed rules and/or exceptions to the rules.

Furthermore, these occurrences and/or inverted attribute indices may be (1) transformed into tagged data records, (2) transformed into a form capable of being ingested into a persistent archive, and (3) transformed into a form capable of being instantiated onto a query-able mechanism.

Each of the foregoing methods may be tangibly embodied as a series of processor-executable instructions stored on a processor-readable medium. Again, for purposes of this disclosure, a “medium” is any processor accessible device capable of storing data, including without limitation RAM, ROM, EPROM, EEPROM, PROM, disk, floppy disk, hard disk, CD-ROM, DVD, flash memory, etc.

III. Third Embodiment

A. Knowledge-Based Persistent Archives With At Least One Self-Describing, Infrastructure-Independent Or Executable Specification

In a third embodiment of the invention, a knowledge-based persistent archive which includes at least one self-describing, infrastructure-independent or executable specification is provided. Referring to Figure 46, in this embodiment, the archive comprises at least one representation of the collection of data objects, identified with numeral 4600; at least one self-describing, infrastructure-independent or executable specification of one or more transformations relating to the collection, identified with numeral 4602; and at least one self-describing, infrastructure-independent or executable specification of one or more rules encoding knowledge relevant to the collection, identified with numeral 4604. Optionally, the archive includes a self-describing, infrastructure-independent representation of a presentation mechanism for one or more of the data objects in the collection.

Referring to Figure 47, an ingestion network 4720 is illustrated. This network represents all the possible states of the data objects as they are transformed from the

form in which they were received from the producer 4700 of the data into a form suitable for ingestion into the archive 4718, into a form suitable for presentation to a consumer 4716, into a form suitable for instantiation onto a query-able mechanism, and into a form suitable for migration onto a new medium. The transitions between the states represent the transformations that the data objects undergo.

In this ingestion network, submission information packages (SIPs) are received from producer 4700 and form the initial representation of the collection in state s_0 which is identified by numeral 4702. Typically, this data is in the form of raw data records.

The data in state s_0 then undergoes a transformation t_1 to form the data in state s_1 , identified by numeral 4704. In the example illustrated, the data in state s_1 is assumed to be in a form ready for archiving to archival storage 4718. Thus, this data may be referred to as an archival information package (AIP).

The transformation t_2 is assumed to result in data which is lossy, and therefore is unusable. That is why there are no transitions out of the state s_2 identified by numeral 4722.

The transformation t_3 transforms the data in state s_1 to state s_3 , which is identified by numeral 4706. Similarly, the transformation t_4 transforms the data in state s_3 to state s_4 , which is identified by numeral 4708. This data is assumed to be in a form ready for archiving to archival storage 4718, and thus may be referred to using the AIP nomenclature.

The transformation t_5 transforms the data in state s_4 to state s_5 , which is identified by numeral 4710. This data as well is assumed to be in a form ready for archiving to archival storage 4718, and thus may also be referred to using the AIP nomenclature.

The data in state s_4 may also be transformed into a dissemination package (DIP) in state s_6 , which is identified with numeral 4714. The DIP is in a form ready for dissemination to consumer 4716, either through presentation to the consumer, or by instantiating it onto a query-able mechanism, at which point it may be queried by
5 the consumer.

Similarly, the data in state s_5 may be transformed into a dissemination package (DIP) in state s_7 , which is identified by numeral 4712. Again, the DIP is in a form ready for dissemination to consumer 4716, either through presentation to the consumer, or by instantiating it onto a query-able mechanism, at which point it may
10 be queried by the consumer.

The process of migrating the archived data to a new medium may also be represented in the ingestion network. More specifically, migration may be represented as the process of retrieving data from archival storage 4718 and inputting it to an ingestion network at state s_0 . This step is identified by numeral 4724 in Figure
15 47.

Turning back to Figure 46, the representation of the collection 4600 may be any one of the representations of the collection within ingestion network 4720, including the data in any of the states s_0 , s_1 , s_3 , s_4 , s_5 , s_6 , and s_7 . Moreover, there may be more than one of these representations present in the archive. Multiple
20 representations of the collection introduces redundancy into the archive, and thus helps ensure that the content will be preserved.

For example, a representation may be the initial data provided by the producer 4700. Or, it may be a self-describing, infrastructure-independent form of this data, such as the initial data records after being tagged with attribute or element names.
25 The representation may also be data in a form capable of presentation to the

consumer, data in a form capable of being instantiated onto a query-able mechanism, or data in a form capable of being migrated onto a new medium. Or it can be occurrences of attribute or element values, or one or more inverted attribute indices, as described in the previous section. It can also be any representation of a knowledge
5 base, such as a topic map. Or it can be any combination of the foregoing, such as a combination of the raw data records and the data records tagged with attribute or element names.

Referring again to Figure 46, the one or more transformations 4602 included in the archive can be any of the transformations t_1 , t_3 , t_4 , t_5 , any of the transformations
10 between AIPs and DIPs, i.e., between states 4708 and 4714, and between states 4710 and 4712, and any transformation required to place the archived data into a form for migration onto a new medium, as indicated by identifying numeral 4724. Moreover, there may be more than one transformation included in the archive. As with the representations of the collection, multiple transformations introduces redundancy, and
15 thus helps ensure that the content will be preserved.

The one or more transformations 4602 may be content-preserving and therefore invertible. They may also be configured to produce (1) data objects in a form suitable for ingestion into the archive, (2) data objects in a form suitable for instantiation onto a query-able mechanism, (3) data objects in a form suitable for
20 presentation, or (4) data objects in a form suitable for migration onto a new medium. They may also be configured to produce occurrences of attribute or element values, or one or more inverted attribute indices, as described previously. They may also be configured to produce a knowledge base, such as a topic map. Or they may include or comprise any combination of the foregoing.

25 The transformations are expressed in a (1) self-describing, infrastructure-independent, or (2) executable form. (These requirements are expressed in the

disjunctive for the reasons stated in Section III.C.) In one example, the transformations are expressed in the form of Perl scripts. Also, a self-instantiating archive is possible in which the one or more transformations are configured, upon execution thereof by an appropriate processor, system or engine, to automatically transform one of the representations of the collection stored with the archive into a form ready for instantiation onto a query-able mechanism, or presentation to a consumer.

Furthermore, a representation 4600 of the collection may be a product of one of the transformations 4602, or may be an input to one of the transformations 4602. If multiple representations are included, one of the representations may be the input to a transformation, and another may be the product of the transformation.

Referring back to Figure 46, the one or more rules 4604 may be rules (and any valid exceptions) useful for validating the collection at any time. The rules are expressed in a (1) self-describing, infrastructure-independent, or (2) executable form, such as Perl scripts, or F-logic. (Again, these requirements are expressed in the disjunctive for the reasons stated in Section III.C.) A self-validating archive is possible in which the one or more rules may be retrieved and, upon execution by an appropriate processor, system or engine, automatically validate the collection, i.e., determine that the one or more representations 4600 are consistent with the rules and valid exceptions.

The one or more rules may bear a relationship to a DTD discussed previously. A DTD may be viewed as an embodiment of rules and constraints between attributes and elements. These rules and constraints may simply be expressed in declarative form to become the one or more rules 4604 stored with the archive.

Several examples of a mapping between DTD statements and corresponding rules expressed in F-logic are illustrated in Figure 48. Numeral 4800 refers to the rules used to implement the DTD statement `<!ELEMENT X (Y,Z)>`, which specifies a parent element X which has two and only two child elements Y and Z. As shown, this DTD statement implies the following rules which can be applied to determine if a tagged data object conforms to the DTD: (1) return false if the first child is not Y; (2) return false if the second child is not Z; (3) return false if there are no children; and (4) return false if there are other children besides Y and Z.

Numeral 4802 refers to the rules used to implement the DTD statement `<!ELEMENT X (Y|Z)>`, which specifies a parent element X which has one and only one child element which in turn may be either Y or Z. As shown, this DTD statement implies the following rules which can be applied to determine if a tagged data object conforms to the DTD: (1) return false if there is a first child other than Y or Z; (2) return false if there are no children; and (3) return false if there is a child other than a first child.

Numeral 4804 refers to the rules used to implement the DTD statement `<!ELEMENT X (Y)*>`, which specifies a parent element X which has zero or more instances of Y as child elements. As shown, this DTD statement implies the following rule which can be applied to determine if a tagged data object conforms to the DTD: return false if there is a child other than Y.

In one example, a self-validating, self-instantiating knowledge-based persistent archive is formed from the Senate Legislative Activities (SLA) collection described previously in Section II.A. To summarize, this collection represents the activities of Senators during the 106th Congress. A legislative activity can be either a bill, amendment, or resolution (BAR). The files in the collection are organized by Senator, and each file sets forth the legislative activities for that Senator.

An ingestion network 4918 for this example is illustrated in Figure 49. A first transformation from state s_0 (numeral 4900) to state s_1 (numeral 4902) occurs outside the ingestion network. According to this transformation, SLA files in the form of Microsoft Word (file suffix .DOC) are transformed into Microsoft Rich Text Format (file type .RTF) files according to the accessioning policies of the producer.

The files are organized as follows: a Header section identifies Senator name (e.g., “Paul S. Sarbanes”), state (e.g., “Maryland”), reporting period (e.g., “January 6, 1999 to March 31, 2000”), and reporting entity (“Senate Computer Center Office of the Sergeant at Arms and Committee on Rules and Administration”); Section I sets forth Sponsored Measures; Section II, Cosponsored Measures; Section III, Sponsored Measures Organized by Committee Referral; Section IV, Cosponsored Measures Organized by Committee Referral; Section V, Sponsored Amendments; Section VI, Cosponsored Amendments; and Section VII, Subject Index to Sponsored and Cosponsored Measures and Amendments.

Sections III and IV contain the same BARs as Sections I and II, but grouped by committee referral (e.g., “Senate Armed Services” and “House Judiciary”). Section VII contains a list of subjects with references to corresponding BAR identifiers: “Zoning and zoning law → S 9, S.Con.Res.10, S.Res.41, S.J.Res.39”. A measure can be any of the BAR types, i.e., a bill, amendment, or resolution. A resolution can be simple, joint, or concurrent. Initially, the following fourteen data field are identified for extraction and tagging: abstract, bar_id, committee, congressional_record, cosponsors, date_introduced, digest, latest_status, official_title, sponsor, statement_of_purpose, status_actions, submitted_by, submitted_for. The initial collection contains 99 files, representing the activities of 99 Senators.

Referring to Figure 49, the .RTF files then enter the ingestion network 4918. A transformation is first attempted according to which the files are transformed into

.HTML files at state s_2 (numeral 4904). However, this transformation drops Sections III and IV, and is not continued since it is lossy and clearly not content-preserving.

Next, the .RTF files are transformed into tagged XML files using an rtfxml module and OmniMark, a stream-oriented, rule-based data extraction and programming language. This transformation is represented in the figure as the transformation from state s_1 to s_3 (numeral 4906).

The transformation from s_3 to s_4 (numeral 4908) is the main wrapping step used to extract and tag the files with the fourteen initially defined data fields. In addition to tagging attributes, this step also tags occurrences of the attributes. To perform occurrence tagging, the Perl script of Figures 36A-36Z is used. The output of this transformation is a flat file of occurrences of the data field (=attribute) values. Each occurrence is expressed as an oav 3-tuple (occurrence, attribute, value) where an occurrence in turn is expressed in the form of the 2-tuple (filename (=senator_id), line number). Thus, assuming the attribute 'date_introduced' shows up in the file for Senator Paul Sarbanes (senator_id = 106) at line 25 with a value 1/19/1999 and at line 55 with a value 3/15/2000, the following 3-tuples result: ((106, 25), 'date_introduced', '1/19/1999) and ((106,55), 'date_introduced', '3/15/2000').

An additional transformation occurs from state s_4 to s_4 . This transformation is identified with numeral 4910. In this transformation, some of the initial candidate attributes may be decomposed further to capture all the relevant information content, or they may be redefined to ensure that all attributes are populated. For example, as a result of this process, the initial attribute 'list_of-sponsors' is further decomposed as follows: list_of_sponsors \rightarrow (sponsor) and sponsor \rightarrow (name, date). At the conclusion, of this process, closure of the attribute selection process may be declared (see Figure 43 and related discussion).

As part of this process, the occurrences could be converted into a XML preservation format and queried using XMAS, XQL, or QUILT (an emerging XML standard). Or, they could be converted into a relational model, and queried using SQL queries. These queries may be used as part of the attribute closure process. For
5 example, the attribute 'abstract' is determined to be empty, and thus a candidate for dropping.

The oav tuples can also be inverted to form inverted attribute indices. In one example, the oav tuples are inverted to form Prolog assertions which can be used for completeness checking. For example, the oav ((105,20), senator, 'RICHARD G.
10 LUGAR of INDIANA') could be inverted to form the Prolog assertion senator(105,20,'RICHARD G. LUGAR of INDIANA').

The transformation from s_4 to s_5 (numeral 4916) builds the desired archival information packages (AIPs) in XML. The content and structure of the original SIPs is preserved by assembling data objects from subobjects using the oav tuples. The
15 result of this process is a collection of XML tagged data objects which reflect the DTD illustrated in Figure 37.

The transformation from s_4 to s_6 (numeral 4912) creates a consolidated version of the collection. The desirability of consolidation is apparent from an analysis of the oav tuples which reveals that there are 44,145 occurrences of BARs, yet there are only
20 5,632 distinct BARs. To perform this consolidation, the collection is reverse engineered to create a database of 5,632 BARs. The SLA collection may then be re-expressed as a particular view of this database in which the individual BARs are grouped by Senator.

As part of this consolidation transformation, integrity checks can be performed
25 to ensure completeness of the collection. For example, through execution of the rules

portion of the archive, the collection could be checked to ensure that there is file for each Senator appearing somewhere in the collection. The analysis reveals that there are three Senators for which corresponding files do not appear: John Chafee of Rhode Island, Phil Gramm of Texas, and Zell Miller of Georgia. To handle this condition,
5 an exception is created to the rule indicating that each of the Senators appearing in the collection must have their own file. This exception specifies that individual files need not appear for the three Senators listed above. This exception then becomes a rule when it is added to the rules portion of the archive. This allows collection validation and integrity to be declared even when individual files for the three Senators are not
10 present.

The transformation from s_4 to s_7 (numeral 4914) creates a topic map version of the collection. This topic map provides a map between concepts and attributes.

An example of a topic map expressed in XML for the SLA collection is illustrated in Figures 54A-54C. The format of the topic map conforms to an emerging
15 Web-based standard under development by XTM, the standards body for the web-based standard. This standard has its genesis in the ISO topic map standard. For more information on the XTM web-based standard, the reader is referred to www.xtm.org.

The purpose of the topic map is to define the semantics of the collection. In
20 the example of Figures 54A-54C, these semantics are defined in terms of topics and relationships between the topics. In addition, in the example illustrated, the semantics are defined in terms of links between topics and occurrences of these topics in the XML-tagged documents representing bills, amendments, and resolutions.

The first portion of the topic map is an embedded DTD defining the structure
25 of the topic map. In this example, the topic map consists of topics and/or

associations. A topic has a name and links to occurrences of that topic in the underlying tagged documents. A link has two attributes. The first is the role played by the topic in the occurrence. The second is a physical link to the occurrence of the topic.

5 Following the embedded DTD is an XML document with a listing of the occurrences of four topics which have been tagged: First, there is a listing of occurrences of the topic t1 (Apartment houses). The topic "Apartment houses" has an occurrence in the Senate bill S.463. The role "Discussed In" indicates that the topic is discussed in the bill S.463.

10 Second, there is a listing of the occurrences of the topic t2 (Children). This listing indicates that the topic "Children" is discussed in the following bills and resolutions: S.300, S.463, S.1638, S.1673, S.1709, S.Res.125, and S.Res.258.

15 Third, there is a listing of the occurrences of the topic t3 (Welfare). This listing indicates that the topic "Welfare" is discussed in the following bills and resolutions: S.463, S.1277, S.1709, S.Con.Res.28, S.Res.125, and S.Res.260.

 Fourth, there is a listing of the occurrences of the topic t4 (Youth employment). This listing indicates that the topic "Youth employment" is discussed in the Senate bill S.463.

20 As indicated in the embedded DTD, an association has a type, and may have one or more rules (element "assocrl"). Each such rule has two attributes. The first is the role of the topic in the association, and the second is a link to one of the other topics or an occurrence of the topic in the tagged data objects.

 Following the listings, there is a section which identifies associations relevant to the collection. In the particular example illustrated, two associations are identified.

The first is an association of topics which all appear together in one and only one bill. All four topics are identified in this association, since each appears together in one and only one bill: S.463. The second is an association of topics which all appear together in two or more bills. In this association, two topics are identified, t2
5 (Children) and t3 (Welfare), since both appear together in three bills and resolutions: S.463, S.1709, S.Res.125. In lay terms, the first association attempts to identify those topics that are only very loosely related, while the second attempts to identify those topics that are more closely related.

A topic map such as illustrated in Figures 54A-54C may serve a variety of
10 purposes. First, it can express hidden information and relationships about the collection, which can be useful when the archive is re-instantiated. For example, this information may be helpful for purposes of understanding the collection or issuing queries against it. Second, it may be used to create different views of the archive for different audiences, e.g., a researcher vs. an archivist. Third, it may be helpful for
15 purposes of identifying rules and constraints that may exist in relation to the topics and data objects. These rules and constraints provide additional context which may be helpful for purposes of understanding and validating the collection. For example, if a relationship is found to exist between topics A and B, then the topic map could specify constraints and rules that must be satisfied by topic A. These rules and
20 constraints could be added to the topic map as well, and therefore, extend the knowledge base of the archive. Fourth, the topic map can embody the knowledge base of any of the persistent archives which are the subjects of Sections II or III of this disclosure.

Note that the dissemination information packages (DIPs) may be formed from
25 the database resulting from the transition to state s_6 (numeral 4912) and the topic map resulting from the transition to state s_7 (numeral 4914). The database represents a form of the collection which may be instantiated onto a query-able mechanism such

as a relational database, or an XML database such as TAMINO. The topic map represents a form of the collection which may be presented to a consumer.

At the conclusion of this process, one or more representations of the collection suitable for archiving can be declared. Any of the intermediary or final products referred to above could form a collection representation suitable for archiving. Candidates include the original .RTF files, the XML tagged files from state s_5 (numeral 4916) coupled with the occurrence tuples from state s_4 (numeral 4908), the BAR database from state s_6 , and the topic map from state s_7 (numeral 4914).

In addition to one or more of the foregoing collection representations, the archive is formed from a self-describing, infrastructure-independent, or executable specification of one or more of the transformations used to create these various collection representations. One example of such a specification is the Perl script of Figures 36A-36Z.

The archive is also formed from a self-describing, infrastructure-independent, or executable specification of one or more rules relevant to the collection. One example is an executable specification of F-logic embodying the rules implied by the DTD of Figure 37 augmented to include any additional rules and/or valid exceptions needed to declare validation of the collection. For example, the rule that a file must appear for each Senator mentioned in the collection coupled with the exception that an individual file need not appear for the three Senators Chafee, Gramm, and Miller could be added to this rules specification.

B. Methods Involving Knowledge-Based Persistent Archives With Executable Specifications

A method of automatically placing one or more data objects from an archived collection into a form suitable for instantiation onto a query-able mechanism is also

provided. Referring to Figure 50, one embodiment of this method comprises, in step 5000, retrieving from the archive a self-describing, infrastructure-independent or executable specification of one or more transformations relevant to the collection; in step 5002, retrieving from the archive a representation of one or more data objects in the collection; and, in step 5004, executing the specification to automatically place the one or more data objects into a form suitable for instantiation onto the query-able mechanism.

Also provided is a method of automatically validating a collection of data objects within a persistent archive. Referring to Figure 51A, one embodiment of this method comprises, in step 5100, retrieving from the archive a self-describing, infrastructure-independent or executable specification of one or more rules relevant to the collection; and, in step 5102, executing the specification to automatically validate the collection.

Referring to Figure 51B, in the method of Figure 51A, the step of validating the collection may be performed by, in substep 5104, producing occurrences of attribute or element values; and, in substep 5106, determining that the occurrences are consistent with the rules encoded by the specification and any valid exceptions.

Referring to Figure 52, a method of automatically presenting one or more data objects from a persistent archive of a collection of data objects is also provided. One embodiment of this method comprises, in step 5200, retrieving from the archive a self-describing, infrastructure-independent or executable specification of one or more transformations relevant to the collection; in step 5202, retrieving from the archive a representation of one or more data objects in the collection; and, in step 5204, executing the specification to automatically place the one or more data objects from the collection in a form suitable for presentation.

A method of automatically placing an archived collection of data objects into a form suitable for migration to a new medium is also provided. Referring to Figure 53, one embodiment of this method comprises, in step 5300, retrieving from the archive a self-describing, infrastructure-independent or executable specification of one or more transformations relevant to the collection; in step 5302, retrieving from the archive one or more data objects from the collection; and, in step 5304, executing the specification to automatically place the collection into a form suitable for migration to a new medium.

C. Systems Involving Knowledge-Based Persistent Archives With Executable Specifications

A system is also provided which includes an engine for executing self-describing, infrastructure-independent, or executable specifications. This system may further include a validation subsystem for validating the collection by commanding the engine to execute at least one self-describing, infrastructure-independent or executable specification encoding one or more rules relevant to the collection. In one example implementation, the engine is part of a deductive database. In another, it is an XSLT engine. In a third example implementation, a single engine is provided that performs ingestion, instantiation, and validation by executing appropriate specifications. For purposes of this disclosure, the term “engine” refers to any mechanism, whether software, hardware, or a combination of hardware and software, that is capable of executing or being built or written to execute one or more of the specifications.

This system may further include a transformation subsystem for transforming one or more data objects in the collection by commanding the engine to execute at least one self-describing, infrastructure-independent, or executable specification of one or more transformations relevant to the collection.

At the time the transformation is to be performed, the one or more transformation specifications should either be (1) self-describing and infrastructure-independent, so that a suitable system or engine for executing the specification can be built or written or so that the specification can be put into a form which is executable
5 by a pre-existing engine or system, or (2) executable, so that the specification may be executed by a pre-existing system or engine.

Similarly, at the time validation is to be performed, the one or more rule specifications should either be (1) self-describing and infrastructure-independent, so that a suitable system or engine for executing the specification can be built or written
10 or so that the specification can be put into a form which is executable by a pre-existing system or engine, or (2) executable, so that the specification may be executed by a pre-existing engine or system.

In one embodiment, a single engine is capable of executing both the one or more transformation specifications, and the one or more rule specifications.

While various embodiments of the invention have been described, it will be apparent to those of ordinary skill in the art that many more embodiments and implementations are possible that are within the scope of this invention.
15

In particular, many alternatives to XML as a tagging language are possible, including SGML (Standard Generalized Markup Language). In general, any tagging
20 format is possible as long as the tagging mechanism is reasonably apparent from the tagged data, and the language can be parsed. For example, the following tagged format in LISP syntax is possible:

(book (author "Jeff")
(title "All's Well That Ends Well")
...)
25

In addition, many alternatives to HTML as a presentation language are possible, including DHTML (Dynamic HTML), XHTML (Extensible HTML), RDF, PDF, etc. Moreover, many alternatives to XSLT as a presentation mechanism are possible. In general, the presentation mechanism should be able to map a representation of a collection or a data object (e.g., an XML DTD) into a presentation language such as HTML, and XSLT, as a scripting language, is a good choice. However, other candidates include scripting languages such as Perl, Python, etc., but any general purpose language could also do.

Also, many examples of query-able mechanisms are possible, including (1) relational databases such as DB2, Sybase, Informix, Illustra; (2) hierarchical databases such as Ariel; (3) XML-based databases such as TAMINO or Excelon; (4) mechanisms for querying tagged documents such as XQuery (the current W3C recommendation), Quilt, a UCSD/SCDS developed language known as XMAS (equivalent to MIX mediator), XPath, XQL, etc., and (5) file systems.

Moreover, many alternatives to DTD and SQL create table commands are possible for the purpose of representing the logical structure of a collection, including XML Schema, RELAX, RDF, RDF-Schema, SOGX, DSP, Schematron, XML-Data, DCB, and Xschema/DDML. In general, any schema language is possible provided it allows for expression of the constraints on the structure of conforming documents or data objects and allows one to distinguish between documents/data objects that conform to the schema, and those that do not.

There are also several possible ways to express topic maps other than through the ISO/IEC 13250 standard. Other examples include a XML Topic Map DTD, or XTM (XML Topic Maps).

There are also several possible ways of specifying a knowledge-base other than through KIF, Prolog, or XTM. Additional examples include DAML+OIL (see www.daml.org) and XOL (XML-Based Ontology Exchange Language). In general, any mode of expression is possible that allows one to express basic relationships, and/or that has an inference mechanism (e.g., Prolog rules) that allows one to derive
5 new relationships from existing ones.

Accordingly, the invention is not to be restricted except in light of the attached claims and their equivalents.